2023 年度 卒業論文

OCaml における Embedding by Unembedding

東北大学 工学部
電気情報物理工学科

C0TB2512 類家 健永

指導教員：松田 一孝 准教授
論文指導教員：松田 一孝 准教授

2023 年 2 月 28 日　23:00–23:30
電子情報システム・応物系 3 号館 2 階 208

要旨

ステキな論文の概要

目次

第 1 章　導入 1

第 2 章　準備 2

2.1　De Bruijn インデックス 2

2.2　HOAS 3

2.3　Embedding by Unembedding 4

第 3 章　結論 5

謝辞 6

参考文献 7

ステキな付録 8

第 1 章

導入

領域特化言語は特定の問題を解決するために、その問題の領域に特化した言語である。特に他の言語のライブラリの形で実装された領域特化言語は埋め込み領域特化言語と呼ばれる。埋め込み領域特化言語はその実装方式から、他の言語のユーザーがホスト言語の機能やエコシステムを利用可能であるという利点がある。特に、バインディング機能は実装が面倒であるため、埋め込み領域特化言語の利点を得て実装することが望ましい。バインディング機能を埋め込むための方法として、HOAS が挙げられる。HOAS はバインディング機能をホスト言語の高階関数を利用して表現する。HOAS はユーザが利用しやすい一方、双方向変換や漸増計算などの複雑な意味の言語を扱うことが難しい。

埋め込み領域特化言語の実装方式の 1 つに Embedding by Unembedding（EbU）がある。EbU はユーザが扱いやすい HOAS と環境管理システムを接続するフレームワークであり、双方向変換や漸増計算などの複雑な言語を表現可能にする。元の EbU は Haskell で実装されている。しかし、EbU は Haskell の型クラス、GADT、型族、高階多相などの高度な機能が用いられているため、他の関数型プログラミング言語での実装方式は明らかではなかった。

本研究では OCaml における EbU の実装方式を示す。OCaml は GADT を持つが、型族や型クラス、高階型オペレータを直接表現することができない。そのため、元の EbU で GADT を利用している箇所はそのままに、型族や型クラス、高階型オペレータを利用している箇所については module/functor 機能を用いる。そして、双方向変換の例を通して、元の EbU と比較を行い、その有用性を評価する。

本研究は以下の貢献を行っている。

- EbU を OCaml で実装するときの問題点の特定（3.1）
- Haskell の高度な機能に頼らない、新しい実装方式の提案（3.3）
- OCaml で実装された EbU をユーザが扱いやすいように、利便性の高い API を提供（3.4）
- 複雑な言語を用いたケーススタディを実施し、Haskell での実装と同等の表現力を持つことを確認（4.2）

また、結論（6 章）を述べる前に、関連研究（5 章）について議論を行う。

第 2 章

準備

本章では、EbU を実装するにあたり、その基礎となる知識の準備を行う。EbU はユーザーが扱う HOAS フロントエンドと環境管理システムである de Bruijn 項バックエンド間の接着剤として機能する。したがって、本章では HOAS、de Bruijn 項、Embedding by Unembedding について説明する。

3.1 節では de Bruijn インデックスを、3.2 節では HOAS を、3.3 節では Embedding by Unembedding を扱う。

2.1 De Bruijn インデックス

de Bruijn 項は項の表現方法 1 つである。de Bruijn 項は項に出現する変数を名前を介して参照するのではなく、束縛子を直接示すことで参照する。束縛子の指定には自然数が用いられ、変数は束縛子からの深さで表現される。

例えば、型無し λ 計算では恒等関数をラムダ抽象を用いて $\lambda x.\ x$ と表現する。λx は x を指している。したがって、恒等関数では x は深さが 0 であるため、de Bruijn 項では $\lambda.\ 0$ と表現される。

変数が複数ある場合も同様である。$\lambda x.\lambda y.\ x\ (y\ x)$ は $\lambda.\lambda.\ 1\ (0\ 1)$ に対応する。

自由変数を含む項に対しては少し注意が必要である。例えば、関数適用の項は $x\ y$ と表現される。この項はラムダ抽象が含まれないため、そのままでは de Bruijn 項として表現できない。そこで、次の環境をあらかじめ与える。

$$\begin{array}{rcccc} \Gamma & = & x & \mapsto & 4 \\ & & y & \mapsto & 3 \\ & & z & \mapsto & 2 \\ & & a & \mapsto & 1 \\ & & b & \mapsto & 0 \end{array}$$

ラムダ抽象が含まれない項も、この環境に従って表現できる。関数適用の項は 4 3 と表現される。もちろん、自由変数と束縛変数を両方含む項も表現ができ、$\lambda w.\ y\ w$ は $\lambda.\ 4\ 0$ と表現される。ここで、y を示すインデックスが 1 つずれるが、これは束縛子の情報が環境に追加されたために発生する。

2.2 HOAS

組み込みにおいて HOAS は変数バインディングの表現方法の 1 つである。変数バインディングをホスト言語の関数で表す。HOAS はゲスト言語の変数をホスト言語の関数で管理するため、ユーザーが変数の名前管理をする必要がない。また、ゲスト言語はホスト言語のエコシステムを得られるメリットがある。しかし、環境を操作する必要がある複雑なセマンティックについてはその実装が明らかではない。

HOAS を利用した組み込み技術に tagless-final style がある。tagless-final style は shallow embedding を改良した組み込み技術である。

例として、足し算言語 $e ::= n \mid e + e$ を考える。shallow embedding はゲスト言語の各コンストラクタをその意味に対応した関数で表す。足し算言語は

$$\begin{aligned} lit &:: Int \to Int \\ add &:: Int \to Int \to Int \end{aligned}$$

で表される。

shallow embedding はコンストラクタを増やすことに有利ではあるが、解釈を増やすことは不利である。tagless-final style は shallow embedding の利点はそのままに、解釈を増やすことに成功している。tagless-final style では、はじめにゲスト言語の式を

$$\begin{aligned} &guestExp :: (\{-lit-\}\, Int \to a) \to (\{-add-\}\, a \to a \to a) \to a \\ &guestExp\ lit\ add = add\ (lit\ 3)\ (add\ (lit\ 4)\ (lit\ 5)) \end{aligned}$$

という多相型で表す。

多相型を用いることで、guestExp に適切な関数を代入することで構文木をたどらずに解釈を増やせる。例えば、$guestExp\ id\ (+)$ とすることで通常の評価を行うことができ、$guestExp\ show\ (fun\ n\ m \to "("\ ++\ n\ ++\ "+"\ ++\ m\ ++\ ")")$ とすることで式の出力が可能となる。

tagless-final style は HOAS としての利点もある。例として、型なしラムダ計算を HOAS で表す。型なしラムダ計算は HOAS で

$$\begin{aligned} &data\ Exp\ where \\ &\quad App :: Exp \to Exp \to Exp \\ &\quad Lam :: (Exp \to Exp) \to Exp \end{aligned}$$

と表される。しかし、この方法では式を解釈する関数の記述が難しい。例えば、$data\ Val = VFun\ (Val \to Val)$ として、評価関数 $eval :: Exp \to Val$ を定義する場合に評価の中で $Val \to Exp$ という関数が必要となったり、元の計算体系で表現できない式を許すことがあったりする。tagless-final style は、この問題を解決する手法でもある。同様の型なし

ラムダ計算において

$$
\begin{aligned}
&\mathit{class}\ \mathit{Lam}\ e\ \mathit{where}\\
&\quad \mathit{app}\ ::\ e \to e \to e\\
&\quad \mathit{lam}\ ::\ (e \to e) \to e
\end{aligned}
$$

について、

$$
\begin{aligned}
&\mathit{instance}\ \mathit{Lam}\ \mathit{Val}\ \mathit{where}\\
&\quad \mathit{app}\ (\mathit{VFun}\ f)\ x\ =\ f\ x\\
&\quad \mathit{lam}\ k\ =\ \mathit{VFun}\ k
\end{aligned}
$$

のようにして、評価関数を定義できる。

EbU は tagless-final style と Unembedding を組み合わせたフレームワークであるため、tagless-final style は重要な技術である。

2.3 Embedding by Unembedding

第 3 章

結論

謝辞

ステキな論文の謝辞

参考文献

ステキな付録

適当な付録。